LOVELY PROFESSIONAL UNIVERSITY

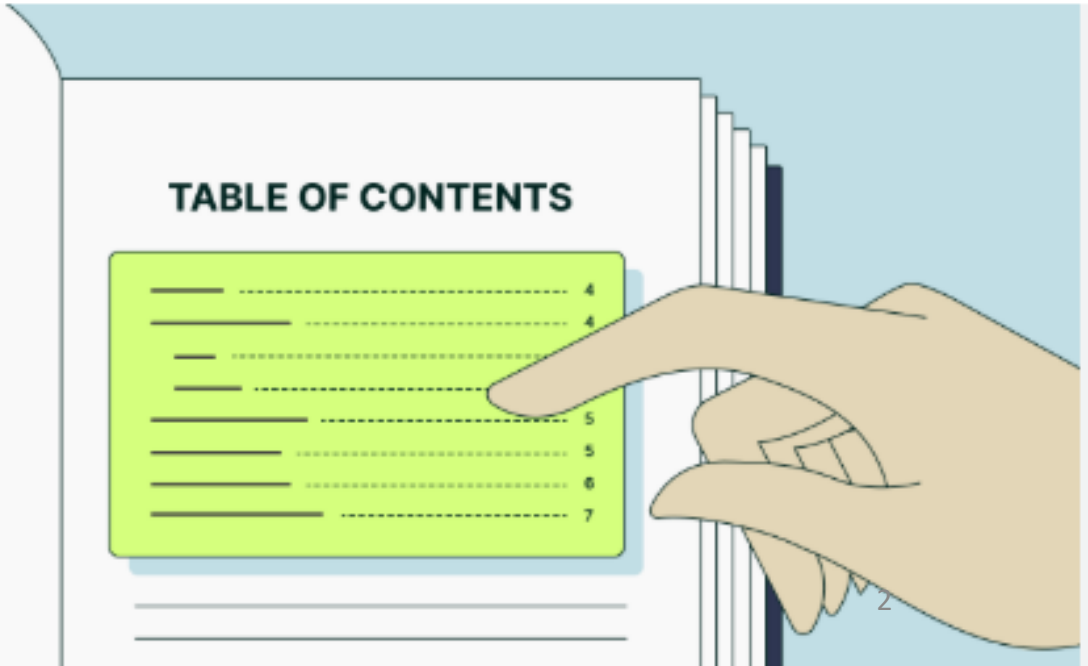
NAAC GRADE A++

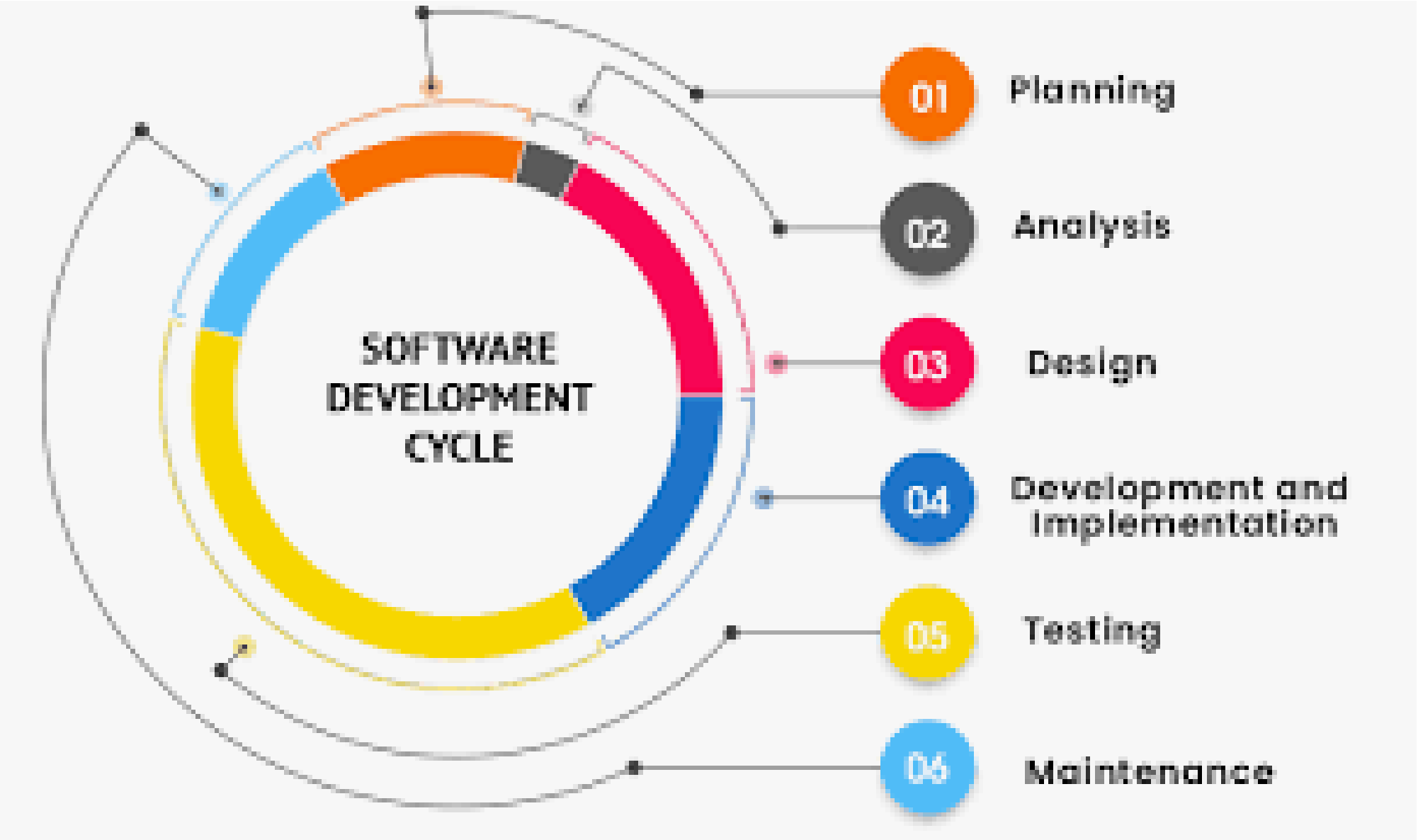
Unit-2

Software Design

Table of Content

- Software design Overview
- Importance of design phase
- Objective of Software design
- Basic issues in software design




SOFTWARE DEVELOPMENT CYCLE
01
Planning
02
Analysis
03
Design
04
Development and Implementation
05
Testing
06
Maintenance

Software Design Overview

- **Definition**: Process of transforming user requirements into a form suitable for coding and implementation.
- **Objective**: Produce a design based on customer requirements documented in the SRS document.
- **Purpose**: Transform the **SRS document** into a **structured design** document.

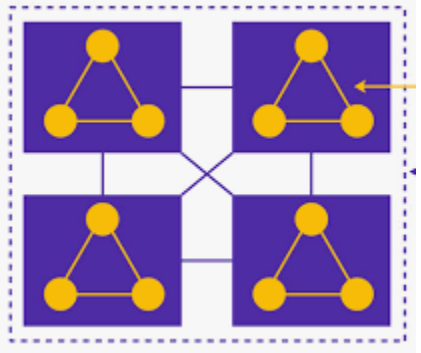
Cont. S/w Design

Design phase transforms SRS document:

- into a form easily implementable in some programming language.

- Software design is the process in which requirements are translated into a – blueprint for constructing a software.

- Initially blueprint shows how the software will look and what kind of data or components will be required to in making it.



Process of s/w Design

- During design process, the s/w specifications are transformed into design models.
- Each design product is reviewed for quality before moving to the next phase of the s/w development
- At the end of the process the design model and specification document is produced.

Importance of the Design Phase

- Provides a clear blueprint for developers.
- Reduces the complexity of implementation.
- Ensures alignment with customer requirements.
- Enhances maintainability and scalability of the software.

Objectives of Software Design

- **Correctness**: Design implements system functionality accurately.
- **Efficiency**: Optimizes time, resources, and costs effectively.
- **Flexibility**: Adapts to changes with minimal effort.
- **Understandability**: Modular design ensures easy understanding
- **Completeness**: Includes all components and interfaces.
- **Maintainability**: Supports easy updates and modifications.
- **Modular Design**: Simplifies debugging and scalability.
- **Documentation**: Provides clarity and ensures future usability.

Levels of Software Design

- **Architectural Design**: Defines overall system structure and components.
- **Conceptual Integrity**: Ensures smooth interaction between components.
- **Solution Domain**: Provides a broad view of design.
- **System Interactions**: Components interact to achieve system goals.

Contd. Levels of Software Design

- **High-Level Design**: Decomposes problems into manageable modules.
- **Control Relationships**: Identifies control flow between modules.
- **Detailed Design**: Examines each module for algorithms, data.
- **Module Specification**: Outcome documented in a detailed specification

Design Specification Models

- **Data Design:** Created by transforming the analysis information model(data dictionary and ERD) into data structures required to implement the s/w.

- **Architectural design:** defines the relationships among the major structured elements of the software, the design patterns

  That can be used to achieve the requirements that have been defined for the system.

  Relationships can be made using UML or USECASE diagrams.

- **Interface Design:** describes how the software elements communicate with each other, with other systems, and with human users; the data flow and control flow diagrams provide the much of the necessary information required

Classification of Design Activity

Design activities are usually classified into two stages:

1. preliminary (or high-level) design
2. detailed design

Preliminary (or high-level) design

- **Outcome** of high level design is called **program structure** or software architecture.
- Through the high level design, a problem is decomposed into a set of modules.

Contd.

2. **Detailed design**

- Focus on module algorithm and structures

- During detailed design each module is examined carefully to design its data structures and the algorithms

Items Designed During Design Phase

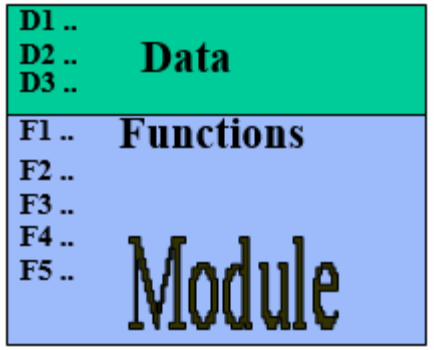
1. module structure

2. control relationship among the modules

3. interface among different modules,
   data items exchanged among different modules,

4. data structures of individual modules,

5. algorithms for individual modules.



Module in Software Design

- **Definition:** A **module** refers to a self-contained unit or component of a program that performs a specific task or set of tasks.
- Modules are designed to break down complex systems into smaller, manageable parts, promoting **modularity** and **reusability**.
- Each module typically has a well-defined interface, making it easier to interact with other modules while encapsulating its internal functionality
- **Encapsulation**: Hides internal details, exposing only necessary functionality.
- **Interface**: Modules interact through defined inputs/outputs (e.g., functions or APIs).
- **Reusability**: Well-designed modules can be reused in different programs.
- **Maintainability**: Easy to update or modify without affecting the whole system.

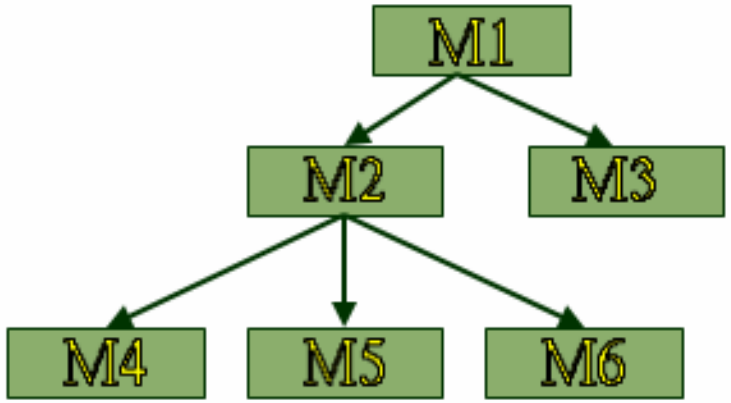
Benefits:

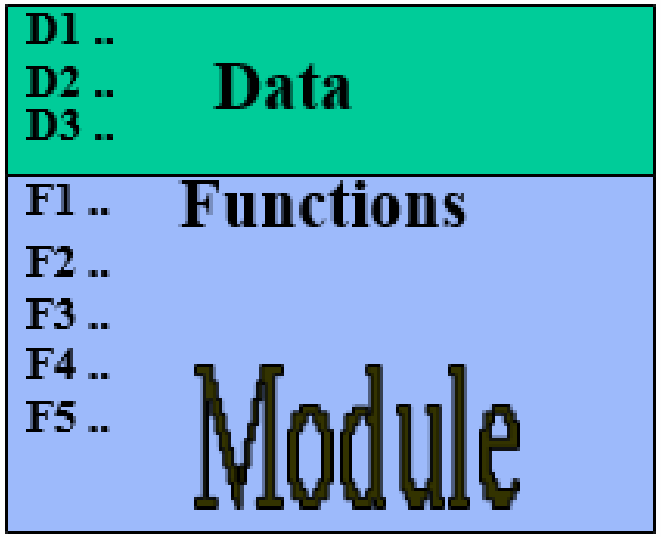
- Simplifies development and testing.
- Increases flexibility and reduces complexity.
- Enhances code maintainability and reusability.

Module structure

A module consists of:

1. several functions
2. associated data structures





Basic issues in software design

- Modularity
- Coupling
- Cohesion

Identify:

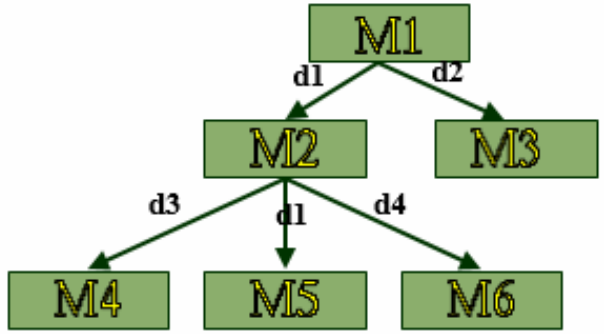
1. Modules
2. control relationships among modules
3. interfaces among modules

Modularity

Modularity is a fundamental attribute of any good design.

- Decomposition of a problem cleanly into modules:
  - Modules are almost independent of each other
  - Follows the divide and conquer principle (tried to reduce the interdependency)

- Software is divided into the different modules or functions. Further we have various attributes. So these modules should have less depende



Contd.

If modules are independent:

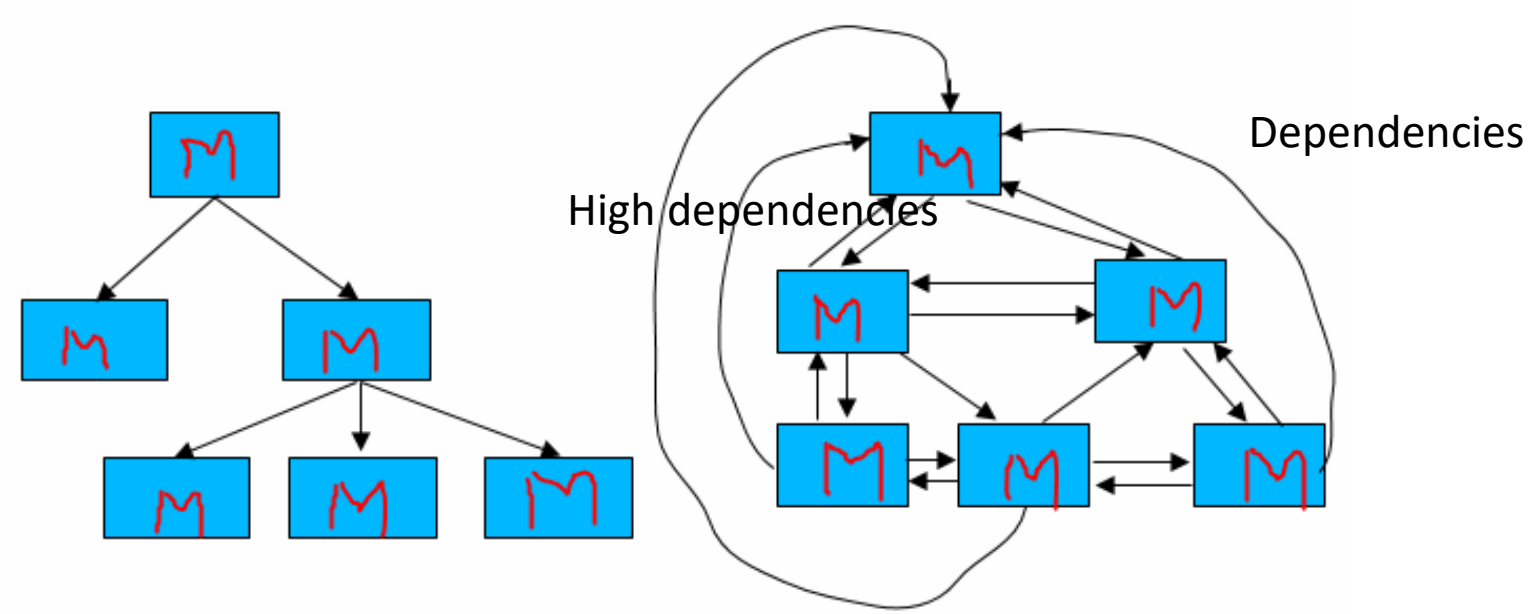
- modules can be understood separately.
- reduces the complexity greatly.

In technical terms, modules should display:

1. high cohesion
2. low coupling

For the better software design.

Example of Cleanly and Non-cleanly Decomposed Modules



Note: reduce the high dependencies using the design techniques

Why modularize a System?

- **Management:** Partition the overall development effort.

- **Evolution:** Decouple parts of system so that changes to one part are isolated from changes to other part

  Principle of Discontinuity(delete a part)

  Principle of Continuity(Addition to the part)

- **Understanding:** Permit system to be understood

  As composition of mind-sized chunks

  With one issue at a time

Advantage of Modularization

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Component with high usage can be re-used
- Concurrent execution can be made possible

Modular design always easy to change, build and maintain

Cohesion (intra module)

- Cohesion represent the *detail design (within module, how elements are interrelating to each other)*
- Cohesion means togetherness or group within the module.
- How we are grouping the various functions/ various methods inside the module.

Coupling (inter module)

- Dependency between two or more modules.
- It measure the degree of interdependence between modules.
- A good software will have low coupling.

A functional independent module has minimal interaction with other modules

Cohesion and Coupling

Cohesion

- Cohesion is intra module.
- It is the measure of functional strength of the modules.
- A cohesive modules performs a single task or function.

Coupling

- Coupling is inter module, between two or more modules.
- A measure of the degree of interdependence or interaction between the two modules.

Types of coupling

Dependency between two or more modules.

- Content Coupling
- Common Coupling
- External Coupling
- Control Coupling
- Stamp Coupling
- Data Coupling

bad/**Worst i.e. least desirable.**

best ……. A good software will have low coupling.

If modules are more inter related means they are highly interdependent to each other.

So debugging or error isolation will become difficult in the particular module.

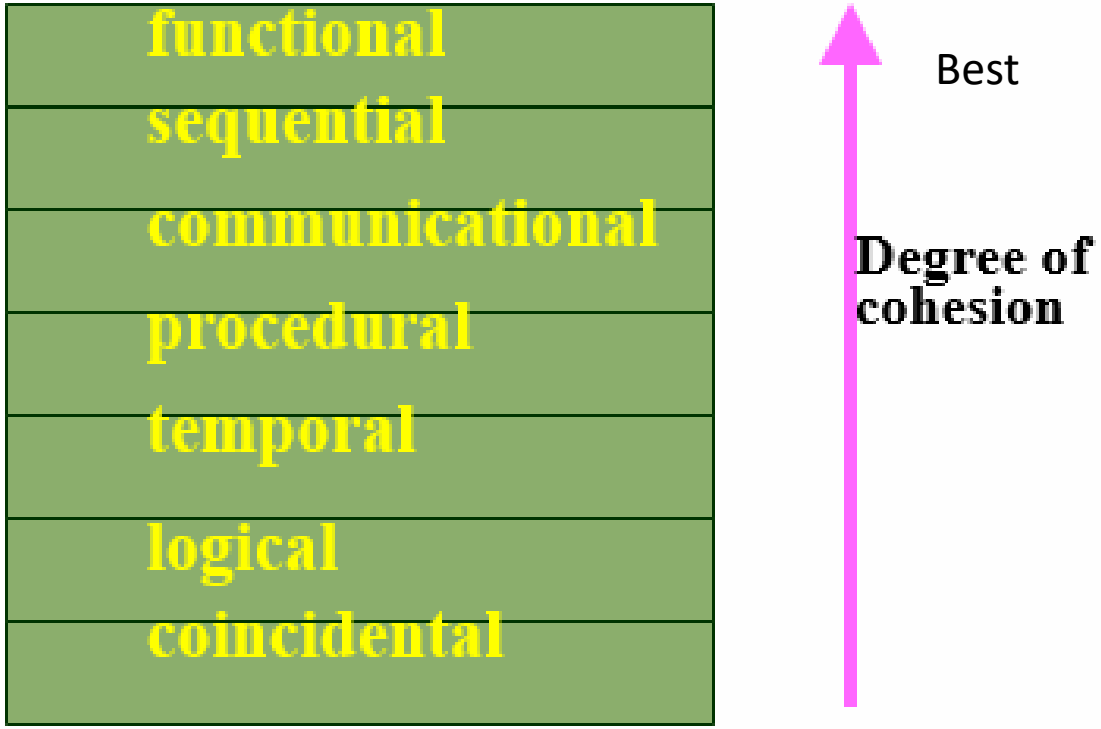
❑How modules become coupled/dependent?

When module share data/ exchange data or they make calls to each other.

❑How to control coupling?

By controlling the amount of information is exchanged between the modules

Types of cohesion



- **Cohesion (intra module)**

- Cohesion represent the *detail design (within module, how elements are interrelating to each other)*
- Cohesion means togetherness or group within the module.
- How we are grouping the various functions/ various methods inside the module.

Quick Test

1. Which document is transformed during the software design phase?

A. Test Plan Document
B. SRS Document
C. Project Charter
D. User Manual

2. What is the primary objective of software design?

A. To debug the software
B. To define the hardware requirements
C. To create a blueprint for implementation
D. To monitor the software process

3. Which level of design identifies the overall structure of the system?

A. Architectural Design
B. High-Level Design
C. Detailed Design
D. Functional Design

4. What is the main focus of detailed design?

A. Identifying system modules
B. Designing algorithms and data structures
C. Establishing control flow among modules
D. Gathering customer requirements

5. Which of the following is NOT an objective of software design?

A. Correctness
B. Efficiency
C. Randomness
D. Maintainability

6. What is the term for designing software that adapts easily to changes?

A. Efficiency
B. Flexibility
C. Completeness
D. Modularity

7. Which issue is addressed during software design?

A. Debugging complex code
B. Identifying control relationships between modules
C. Conducting system testing
D. Writing user manuals

8. What is the outcome of the high-level design stage?

A. Module Specification Document
B. Test Plan Document
C. Program Architecture
D. Requirements Analysis Document

9. What design principle supports easy updates and maintenance?

A. Documentation
B. Efficiency
C. Maintainability
D. Completeness